Accelerating Pattern Matching or How Much Can You Slide?

Ofir Pele and Michael Werman

School of Computer Science and Engineering
The Hebrew University of Jerusalem
{ofirpele,werman}@cs.huji.ac.il

Abstract. This paper describes a method that accelerates pattern matching. The distance between a pattern and a window is usually close to the distance of the pattern to the adjacement windows due to image smoothness. We show how to exploit this fact to reduce the running time of pattern matching by adaptively sliding the window often by more than one pixel. The decision how much we can slide is based on a novel rank we define for each feature in the pattern. Implemented on a Pentium 4 3GHz processor, detection of a pattern with 7569 pixels in a 640×480 pixel image requires only 3.4ms.

1 Introduction

Many applications in image processing and computer vision require finding a particular pattern in an image, *pattern matching*. To be useful in practice, pattern matching methods must be automatic, generic, fast and robust.

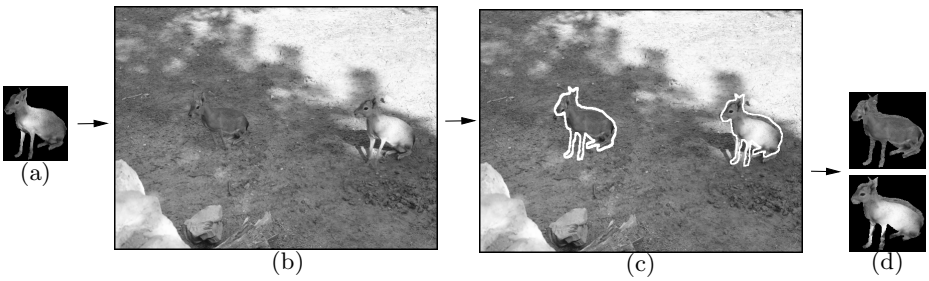

Fig. 1. (a) A non-rectangular pattern of 7569 pixels (631 edge pixel pairs). Pixels not belonging to the mask are in black. (b) A 640×480 pixel image in which the pattern was sought. (c) The result image. All similar masked windows are marked in white. (d) The two found occurrences of the pattern in the image. Pixels not belonging to the mask are in black. The method suggested in this paper reduced the Pele and Werman pattern matching method[1] running time from 21ms to only 3.4ms.

Y. Yagi et al. (Eds.): ACCV 2007, Part II, LNCS 4844, pp. 435–446, 2007.

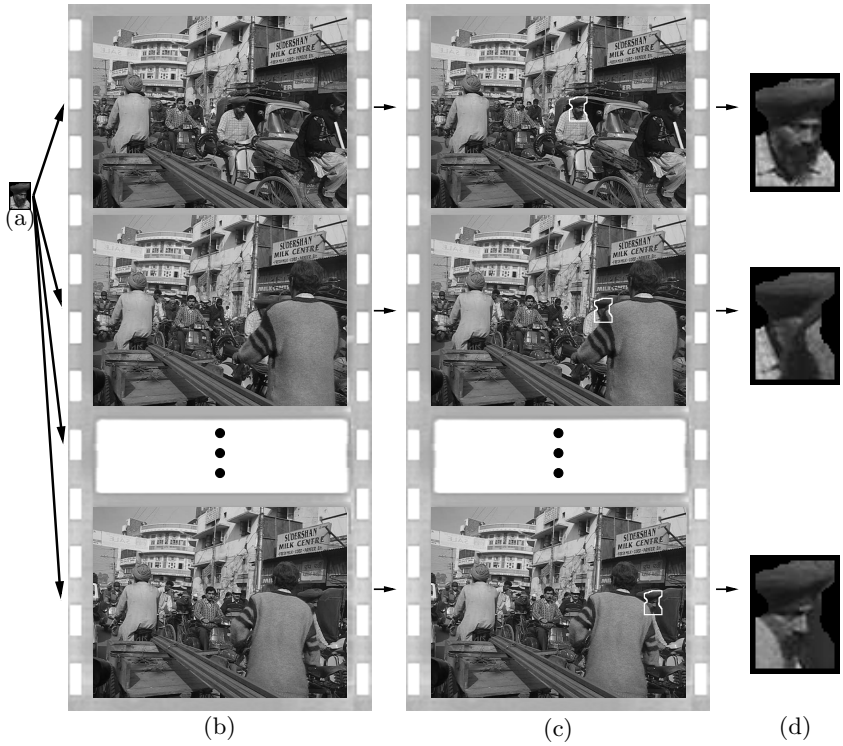

Fig. 2. (a) A non-rectangular pattern of 2197 pixels. Pixels not belonging to the mask are in black. (b) Three 640x480 pixel frames out of fourteen in which the pattern was sought. (c) The result. Most similar masked windows are marked in white. (d) Zoom in of the occurrences of the pattern in the frames. Pixels not belonging to the mask are in black. The method suggested in this paper reduced the Pele and Werman pattern matching method[1] running time from 22ms to only 7.2ms. The average number of samples per window reduced from 19.7 to only 10.6.

Pattern matching is typically performed by scanning the entire image, and evaluating a distance measure between the pattern and a local rectangular window. The method proposed in this paper is applicable to any pattern shape, even a non-contiguous one. We use the notion of "window" to cover all possible shapes.

There are two main approaches to reducing the computational complexity of pattern matching. The first approach reduces the time spent on each window. The second approach reduces the number of windows visited. In this work we concentrate on the second approach.

We suggest sliding more than one pixel at a time. The question that arises is: how much can you slide? The answer depends on the pattern and on the image. For example, if the pattern and the image are black and white checked boards of pixels, the distance of the pattern to the current window and to the next window will be totally different. However, if the pattern is piecewise smooth, the

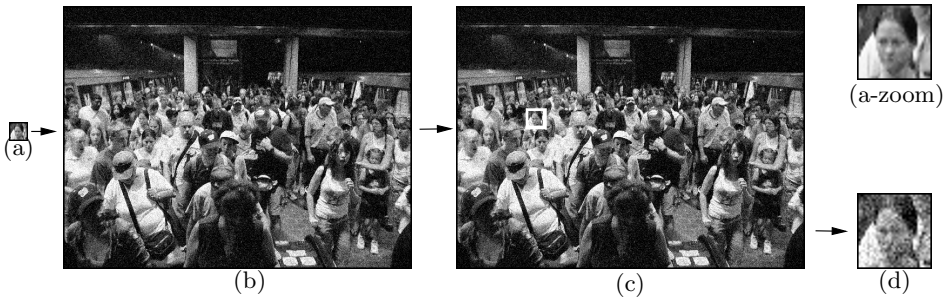

Fig. 3. (a) A rectangular pattern of 1089 pixels. (b) A noisy version of the original 640x480 pixel image. The pattern that was taken from the original image was sought in this image. The noise is Gaussian with a mean of zero and a standard deviation of 25.5. (c) The result image. The single similar masked window is marked in white. (d) The occurrence of the pattern in the zoomed in image. The method suggested in this paper reduced the Pele and Werman pattern matching method[1] running time from 19ms to only 6ms. The average number of samples per window reduced from 12.07 to only 2. The image is copyright by Ben Schumin and was downloaded from: `http://en.wikipedia.org/wiki/Image:July_4_crowd_at_Vienna_Metro_station.jpg`.

distances will be similar. We describe a method which examines the pattern and decides how much we can slide in each step. The decision is based on a novel rank we define for each feature in the pattern.

We use a two stage method on each window. First, we test all the features with a high rank. Most of the windows will not pass and we will be able to slide more than one pixel. For the windows that passed the test we perform the simple test on all the features.

A typical pattern matching task is shown in Fig. 1. A non-rectangular pattern of 7569 pixels (631 edge pixel pairs) was sought in a 640×480 pixel image. Using the Pele and Werman method[1] the running time was 21ms. Using our method the running time reduced to only 3.4ms. All runs were done on a Pentium 4 3GHz processor.

Decreasing the number of visited windows is usually achieved using an image pyramid[3]. By matching a coarser pattern to a coarser level of the pyramid, fewer windows are visited. Once the strength of each coarser resolution match is calculated, only those that exceed some threshold need to be compared for the next finer resolution. This process proceeds until the finest resolution is reached.

There are several problems with the pyramid approach. First, important details of the objects can disappear. Thus, the pattern can be missed. For example, in Fig. 1 if we reduce the resolution to a factor of 0.8, the right occurrence of the pattern is found, but the left one is missed. Using the smaller images the running time decreases from 21ms to 18ms (without taking into account the time spent on decreasing the resolution). Using our approach, both occurrences of the patterns are found in only 3.4ms. Note that smoothness can change

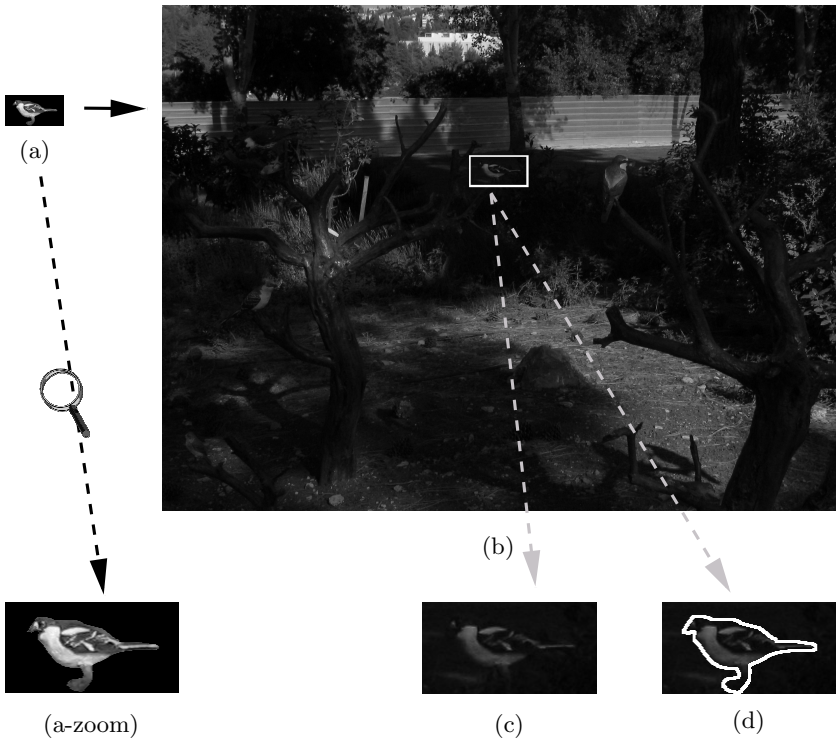

Fig. 4. (a) A non-rectangular pattern of 3732 pixels (3303 edge pixel pairs). Pixels not belonging to the mask are in black. (b) A 2048×1536 pixel image in which the pattern was sought. The area where the pattern was found is marked in white. (c) The occurrence of the pattern in the image zoomed in. (d) The occurrence of the pattern in the image zoomed in, with the exact found outline of the pattern painted in white. The method suggested in this paper reduced the Pele and Werman pattern matching method[1] running time from 437ms to only 51ms. Note the large size of the image. The average number of samples per window reduced from 27 to only 3.7.

between different local parts of the pattern. The pyramid approach is global, while our approach is local and thus more distinctive. The second problem of pyramid approach is the memory overhead.

This paper is organized as follows. Section 2 presents the LU_p rank for pixels and for pairs of pixels. Section 3 describes a method that uses the LU_p rank for accelerating pattern matching. Section 4 presents extensive experimental results. Finally, conclusions are drawn in Section 5.

2 The LU_p Rank

In this section we define a novel smoothness rank for features, the LU_p rank. The rank is later used as a measure that tells us how much we can slide for each

(a)

(b)

Fig. 5. (a) A 115×160 pattern (2896 edge pixel pairs). (b) A 1000×700 pixel image in which the pattern was sought. The most similar window is marked in white. The method suggested in this paper reduced the Pele and Werman pattern matching method[1] running time from 51ms to only 9.2ms. The average number of samples per window reduced from 23 to only 3. The images are from the Mikolajczyk and Schmid paper[2].

pattern. This is first defined for pixels and then for pairs of pixels. Finally, we suggest ways of calculating the LU_p rank.

2.1 The LU_p Rank for Pixels

In this sub-section we use the *Thresholded Absolute Difference* Hamming distance that was suggested by Pele and Werman[1]. This distance is the number of different corresponding pixels between a window and a pattern, where the corresponding pixels are defined as different if and only if their absolute intensity difference is greater than a predefined pixel similarity threshold, q; *i.e.* The distance between the set of pixels, A, applied to the pattern and the current window is defined as (δ returns 1 for true and 0 for false):

$$TAD_A(pattern, window) = \sum_{(x,y)\in A} \delta\left(|pattern(x,y) - window(x,y)| > q\right) \quad (1)$$

We first define the LU rank for a pattern pixel as:

$$\begin{aligned} LU(pattern, (x,y)) &= \max_R \text{ s.t:} \\ \forall \quad 0 \le r_x, r_y \le R \quad & pattern(x,y) = pattern(x - r_x, y - r_y) \end{aligned} \quad (2)$$

Now, if we if we assess the similarity between a pixel in the pattern with an LU rank of R, to a pixel in the window, we get information about all the windows which are up to R pixels to the right and down to the current window. Using this information we can slide in steps of $R + 1$ pixels, without losing accuracy.

The requirement for equality in Eq. 2 is relaxed in the definition of the LU_p rank. In this rank the only requirement is that the absolute difference is not too high:

$$\begin{aligned} LU_p(pattern,(x,y)) = \max_R \ \text{s.t:} \\ \forall \ \ 0 \le r_x, r_y \le R \ \ |pattern(x,y) - pattern(x-r_x, y-r_y)| \le p \end{aligned} \tag{3}$$

Note that the LU and LU_0 ranks for pixels are equivalent.

2.2 The LU_p Rank for Pairs of Pixels

In this sub-section we use the *Monotonic Relations* Hamming distance that was suggested by Pele and Werman[1]. This distance is the number of pairs of pixels in the current window that does not have the same relationship as in the pattern; *i.e.* the basic features of this distance are pairs of pixels and not pixels. Pixel relations have been successfully applied in many fields such as pattern matching[1], visual correspondence[4] and keypoint recognition[5].

Each pattern is defined by a set of pairs of pixels which are close, while the intensity difference is high. We assume without loss of generality that in the pattern the first pixel in each pair has a higher intensity value than the second pixel. The distance between the set of pairs, A, applied to the pattern and the current window is defined as (δ returns 1 for true and 0 for false):

$$MR_A(pattern, window) = \sum_{\substack{[(x_1,y_1),\\(x_2,y_2)]\in A}} \delta\left(window(x_1,y_1) \le window(x_2,y_2)\right) \tag{4}$$

Given a pattern and pair of pixels, $[(x_1,y_1),(x_2,y_2)]$ such that the first pixel has a higher intensity value than the second pixel, *i.e.* $pattern(x_1,y_1) > pattern(x_2,y_2)$, we define the pair's LU_p rank as:

$$\begin{aligned} LU_p(pattern,[(x_1,y_1),(x_2,y_2)]) = \max_R \ \text{s.t:} \ \forall \ \ 0 \le r_x, r_y \le R \\ pattern(x_1-r_x, y_1-r_y) > pattern(x_2-r_x, y_2-r_y) + p \end{aligned} \tag{5}$$

The requirement that the relation must be bigger in at least p is added for stability. Now, if we assess the similarity between a pair of pixels in the pattern with an LU_p rank of R, to a pair of pixels in the window, we get information about all the windows which are up to R pixels to the right and down to the current window. Figure 6 illustrates this. Using this information we can slide in steps of $R+1$ pixels, without losing accuracy.

2.3 How to Calculate the LU_p Rank

We suggest two methods of calculating the LU_p rank of all features (pixels or pairs of pixels) in the pattern. The first is to calculate the rank for each feature. If we denote by $\bar{R}$ the average LU_p rank and by $|A|$ the feature set size, then

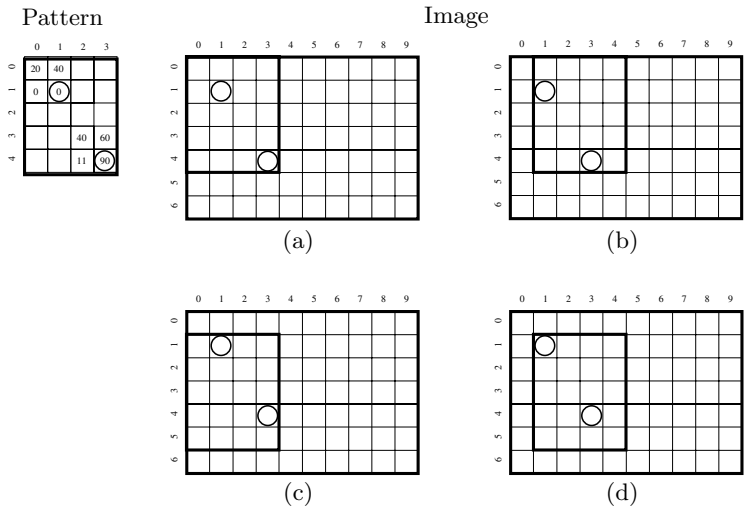

Fig. 6. The pair of pixels in the pattern (marked with two circles): $[(3,4),(1,1)]$, has LU_{10} rank of 1 (Pattern$(3,4) >$ Pattern$(1,1) + 10$ and Pattern$(3,3) >$ Pattern$(1,0) + 10$), etc). Thus, when we test whether Image$(3,4) >$ Image$(1,1)$, we get an answer to these 4 questions (all coordinates are relative to the window's coordinates):
1. In the window of (a), is Window$(3,4) >$ Window$(1,1)$ as in the pattern?
2. In the window of (b), is Window$(2,4) >$ Window$(0,1)$ as in the pattern?
3. In the window of (c), is Window$(3,3) >$ Window$(1,0)$ as in the pattern?
4. In the window of (d), is Window$(2,2) >$ Window$(0,0)$ as in the pattern?

the average time complexity is $O(|A|\bar{R}^2)$. The second method is to test which features have each LU_p rank. This can be done quickly by finding the 2d min and max for each value of R. The Gil and Werman[6] method does this with a time complexity of $O(1)$ per pixel. If we denote by $R_{\max}$ the maximum R value, then the time complexity is $O(|A|R_{\max})$. A combined approach can also be used. Note that the computation of the LU_p is done offline for each given pattern. Moreover, the size of the pattern is usually much smaller than the size of the image; thus the running time of this stage is negligible. In this paper we simply calculate the LU_p rank for each feature.

3 The Pattern Matching Method

The problem of pattern matching can be formulated as follows: given a pattern and an image, find all the occurrences of the pattern in the image. We define a window as a match, if the Hamming distance (*i.e.* Eq. 1 or Eq. 4) is smaller or equal to the image similarity threshold.

In order to reduce the running time spent on each window we use the Pele and Werman[1] *sequential* sampling algorithm. The *sequential* algorithm random samples corresponding features sequentially and without replacement from the

window and pattern and tests them for similarity. After each sample, the algorithm tests whether the accumulated number of non-similar features is equal to a threshold, which increases with the number of samples. We call this vector of thresholds the *rejection line*. If the algorithm touches the *rejection line*, it stops and returns *non-similar*. If the algorithm finishes sampling all the features, it has computed the exact distance between the pattern and the window. Pele and Werman[1] presented a Bayesian framework for sequential hypothesis testing on finite populations. Given an allowable bound on the probability of a false negative the framework computes the optimal *rejection line*; *i.e.* a *rejection line* such that the *sequential* algorithm parameterized with it has the minimum expected running time. Pele and Werman[1] also presented a fast near-optimal framework for computing the *rejection line*. In this paper, we use the near-optimal framework.

The full system we use for pattern matching is composed of an offline and an online part. The offline part gets a pattern and returns the characteristic LU_p rank, two sets of features and the two corresponding *rejection lines*. One set contains all the pattern features. The second set contains all the pattern features from the first set that have an LU_p rank greater or equal to the characteristic LU_p rank.

The online part slides through the image in steps of the characteristic LU_p rank plus one. On each window it uses the *sequential* algorithm to test for similarity on the second set of features. If the *sequential* algorithm returns *non-similar*, the algorithm slides the characteristic LU_p rank plus one pixels right or the characteristic LU_p rank plus one rows (at the end of each row). If the *sequential* algorithm returns *similar* (which we assume is a rare event), the window and all the windows that would otherwise be skipped are tested for similarity. The test is made again using the *sequential* algorithm, this time on the set that contains all the pattern features.

4 Results

The proposed method was tested on real images and patterns. The results show that the method accelerates pattern matching, with a very small decrease in robustness to rotations. For all other transformations tested - small scale change, image blur, JPEG compression and illumination - there was no decrease in robustness. First we describe results that were obtained using the *Thresholded Absolute Difference* Hamming distance (see Eq. 1). Second, we describe results that were obtained using the *Monotonic Relations* Hamming distance (see Eq. 4).

4.1 Results Using the *Thresholded Absolute Difference* Hamming Distance

We searched for windows with a *Thresholded Absolute Difference* Hamming distance lower than $0.4 \times |A|$. The *sequential* algorithm was parameterized using the near-optimal method of Pele and Werman[1] with input of a uniform prior and

a false negative error bound of 0.1%. In all of the experiments, the p threshold for the LU_p rank was set to 5. The characteristic LU_5 rank for each pattern was set to the maximum LU_5 rank found for at least 30 pattern pixels. Note that the online part first tests similarity on the set of pixels with a LU_5 rank greater or equal to the characteristic LU_5 rank. The same relative similarity threshold is used; *i.e.* if the size of this small set is $|A_s|$ we test whether the *Thresholded Absolute Difference* Hamming distance is lower than $0.4 \times |A_s|$. Results that show the substantial reduction in running time are shown in Figs. 2 and 3.

4.2 Results Using the *Monotonic Relations* Hamming Distance

The pairs that were used in the set of each pattern were pairs of pixels belonging to edges, *i.e.* pixels that had a neighbor pixel, where the absolute intensity value difference was greater than 80. Two pixels, (x_2, y_2), (x_1, y_1) are considered neighbors if their l_∞ distance: $\max(|x_1 - x_2|, |y_1 - y_2|)$ is smaller or equal to 2. We searched for windows with a *Monotonic Relations* Hamming distance lower than $0.25 \times |A|$. The *sequential* algorithm was parameterized using the near-optimal method of Pele and Werman[1] with input of a uniform prior and a false negative error bound of 0.1%. In all of the experiments, the p threshold for the LU_p rank was set to 20. The characteristic LU_{20} rank for each pattern was set to the maximum LU_{20} rank found for at least 30 pairs of pixels from the set of all edge pixel pairs. Note that the online part first tests similarity on the set of pairs of edge pixels with a LU_{20} rank greater or equal to the characteristic LU_{20} rank. The same relative similarity threshold is used; *i.e.* if the size of this small set is $|A_s|$ we test whether the *Monotonic Relations* Hamming distance is lower than $0.25 \times |A_s|$. Results that show the substantial reduction in running time are shown in Figs. 1, 4 and 5.

To illustrate the performance of our method, we ran the tests that were also conducted in the Pele and Werman paper[1]. All the data for the experiments were downloaded from `http://www.cs.huji.ac.il/~ofirpele/hs/all_images.zip`. Five image transformations were evaluated: small rotation; small scale change; image blur; JPEG compression; and illumination. The names of the datasets used are *rotation*; *scale*; *blur*; *jpeg*; and *light* respectively. The *blur*, *jpeg* and *light* datasets were from the Mikolajczyk and Schmid paper[2]. *scale* dataset contains 22 images with an artificial scale change from 0.9 to 1.1 in jumps of 0.01; and *rotation* dataset contains 22 images with an artificial in-plane rotation from -10° to 10° in jumps of 1°. For each collection, there were ten rectangular patterns that were chosen from the image with no transformation. In each image we considered only the window with the minimum distance as similar, because we knew that the pattern occurred only once in the image. We repeated each search of a pattern in an image 1000 times.

There are two notions of error: miss detection error rate and false detection error rate. As we know the true homographies between the images, we know where the pattern pixels are in the transformed image. We denote a correct match as one that covers at least 80% of the transformed pattern pixels. A false match is one that covers less than 80% of the transformed pattern pixels. Note

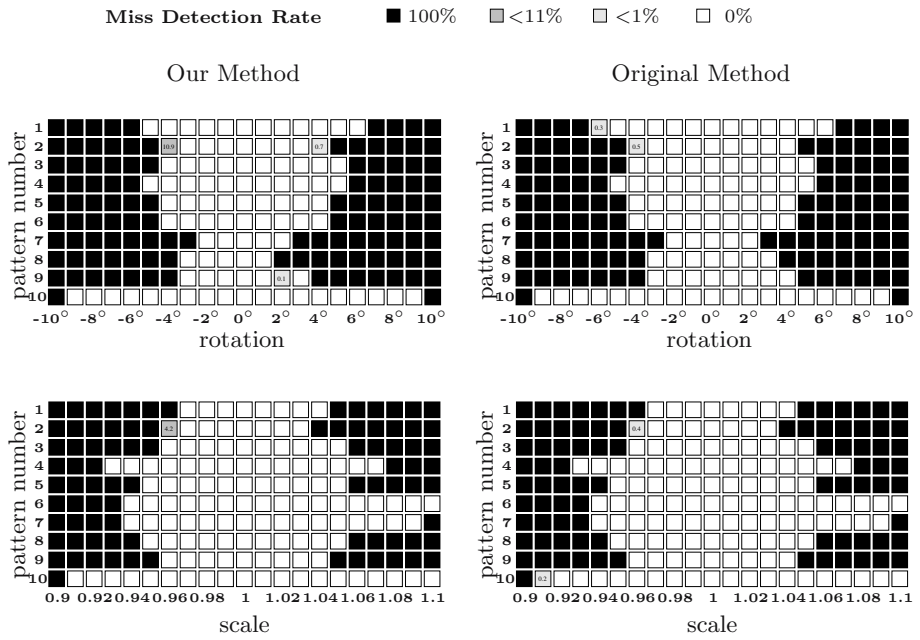

Fig. 7. The miss detection error rates of our method on the *rotation* and *scale* tests. There is a slight decrease in performance with our method on the *rotation* test. On the *scale* test the performance is the same for both methods. Note that our approach runs much faster and that on all other tests (*light*, *jpeg* and *blur*) the miss detections error rates were exactly the same.

that there is also an event of no detection at all if our method does not find any window with a *Monotonic Relations* Hamming distance lower than $0.25 \times |A|$. The miss detection error rate is the percentage of searches of a pattern in an image that does not yield a correct match. The false detection error rate is the percentage of searches of a pattern in an image that yields a false match.

The detection and miss detection error rates were the same as in the Pele and Werman method[1], except in the *rotation* test where there was a slight decrease in performance (see Fig. 7). In the *light* and *jpeg* tests, the performance was perfect; *i.e.* 0% miss detection rate and 0% false detection rate. In the *blur* test, only one pattern was not found correctly in the most blurred image. The miss detection rate and false detection rate for this specific case was 99.6%. In all other patterns and images in the *blur* test, the miss detection rate and false detection rate was 0%. In the *scale* test, there was only one pattern with false detection in two images with scale 0.9 and 0.91. In the *rotation* test, there was only one pattern with false detection in images with rotation smaller than -2° or larger than +2°. Miss detection rates in the *scale* and *rotation* tests (see Fig. 7) were dependent on the pattern. If the scale change or rotation was not too big, the pattern was found correctly.

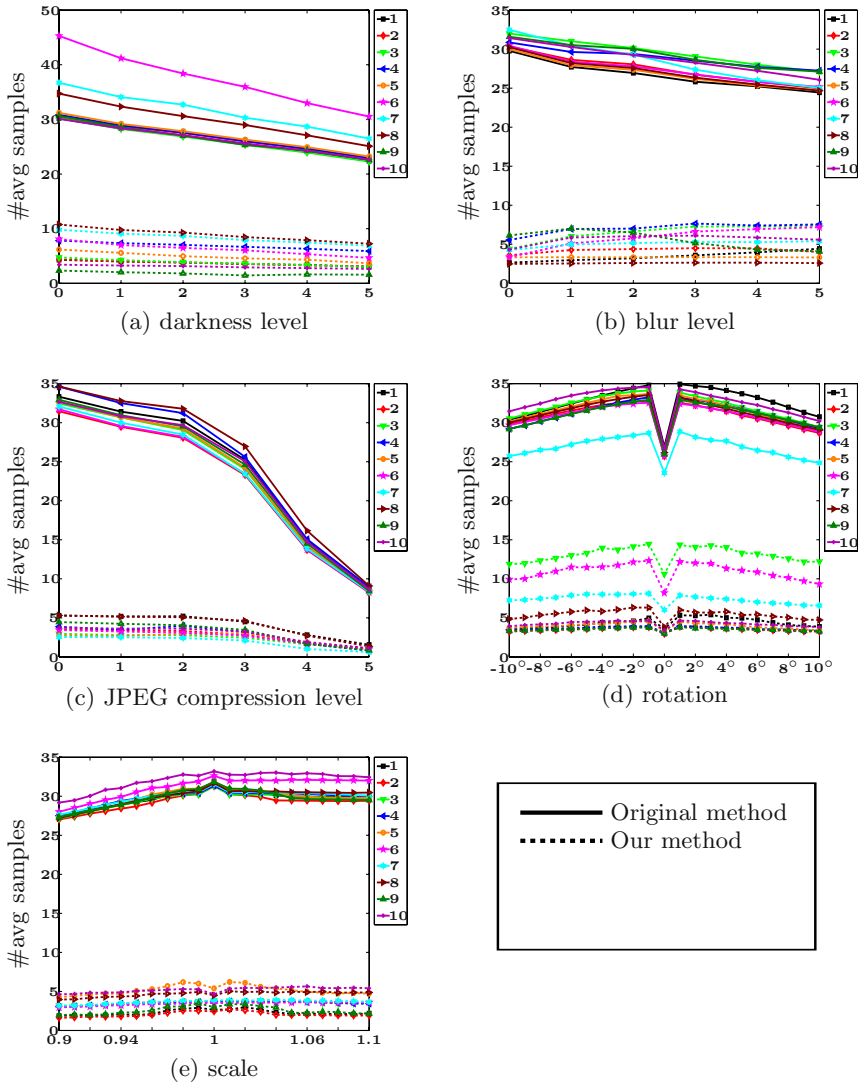

Fig. 8. Average number of samples per window for each of the ten patterns with our method (*dotted lines*) and with the Pele and Werman method[1] (*solid lines*). In all of the tests our approach ran much faster. Note that as our approach skips windows, the average of samples per window can be even smaller than one. It is also noteworthy that the running time in our approach depends on the characteristic LU_{20} rank. For example, in the *light* test - (a), finding pattern number 6 (marked as a *pink star*) using the original method took the most time, while using our method it is one of the patterns that was found the fastest. This can be explained by the fact that this pattern has a characteristic LU_{20} rank of two, which is high compared to the other patterns.

We also measured the average samples taken for each window using our method and the Pele and Werman method (see Fig. 8). In all of the tests our approach ran much faster. Note that as our approach skips windows, the average of samples per window can be even smaller than one. Further, the running time in our approach depends on the characteristic LU_{20} rank.

5 Conclusions

This paper described a method to accelerate pattern matching by adaptively sliding the window often by more than one pixel. We assigned a novel rank to pixels and edge pixel pairs that tells us how many pixels we can slide through the image. We suggested a pattern matching method that uses this rank. Extensive testing showed that the pattern matching was accelerated without losing almost any accuracy.

Faster than real time results were presented, where patterns under large illumination changes, blur, occlusion, gaussian noise, etc, were detected in several milliseconds. To the best of our knowledge, the running time results presented in this paper are the fastest published.

An interesting extension of this work would be to use this novel rank to accelerate additional methods of pattern matching.

References

1. Pele, O., Werman, M.: Robust real time pattern matching using bayesian sequential hypothesis testing. Technical Report 973, Department of Computer Science, The Hebrew University of Jerusalem (2007), `http://www.cs.huji.ac.il/~ofirpele/hs/TR.html`
2. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE Trans. Pattern Analysis and Machine Intelligence 27(10), 1615–1630 (2005)
3. Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)
4. Zabih, R., Woodfill, J.: Non-parametric local transforms for computing visual correspondence. In: Eklundh, J.-O. (ed.) ECCV 1994. LNCS, vol. 800, pp. 151–158. Springer, Heidelberg (1994)
5. Lepetit, V., Fua, P.: Keypoint recognition using randomized trees. IEEE Trans. Pattern Analysis and Machine Intelligence 28(9), 1465–1479 (2006)
6. Gil, J., Werman, M.: Computing 2-d min, median, and max filters. IEEE Trans. Pattern Analysis and Machine Intelligence 15(5), 504–507 (1993)